Tampereen yliopisto

Roope Karvinen

AUTOMATISOIDUN KAUPANKÄYNTIJÄRJESTELMÄN SUUNNITTELU

Informaatioteknologian ja viestinnän tiedekunta
Kandidaatintyö
Kesäkuu 2019

TIIVISTELMÄ

Roope Karvinen: Automatisoidun kaupankäyntijärjestelmän suunnittelu
Tampereen yliopisto
Ohjelmistotekniikka
Kandidaatintyö
Kesäkuu 2019

Algoritminen kaupankäynti tarkoittaa tietokoneilla käytävää automaattista kaupankäyntiä, jossa tietokone hyödyntää erilaisia algoritmeja ja sijoitusstrategioita tuottojen saavuttamiseen. Tässä kandidaatintyössä käsitellään algoritmista kaupankäyntiä ja suunnitellaan siihen soveltuva automatisoitu järjestelmä. Työ vastaa kysymykseen: ”miten suunnitellaan algoritmiseen kaupankäyntiin soveltuva ohjelma?”. Luvuissa kaksi ja kolme käydään läpi teoriaa kaupankäynnistä ja ohjelmistojen suunnittelusta. Luvussa neljä nämä aiheet yhdistetään ja suunnitellaan automaattista kaupankäyntijärjestelmää.

Työn tuloksena on suunnitelma, jonka perusteella pitäisi pystyä ohjelmoimaan kyseinen ohjelma. Suunnitelmasta löytyy komponenttikaavio komponenteista, joita ohjelmassa tarvitaan sekä joitain komponentteja on suunniteltu myös tarkemmin luokkatasolla. Suunniteltaessa järjestelmää on pyritty hyödyntämään ohjelmistojen suunnitteluperiaatteita, suunnittelumalleja sekä olio-ohjelmoinnin periaatteita.

Avainsanat: Algoritminen kaupankäynti, suunnitteluperiaate, suunnittelumalli, kaupankäynnin automatisointi, ohjelmistojen suunnittelu

SISÄLLYSLUETTELO

KUVALUETTELO

LYHENTEET JA MERKINNÄT

BB	Bollinger bands
Current ratio	Yrityksen maksuvalmius noin vuoden aikajänteellä
EPS	Earnings per share
ETF	Exchange Traded Fund
MA	Moving Average
MACD	Moving Average Convergence Divergence
P/E	Price/Earnings
P/S	Price/Sales
RSI	Relative Strength Index
ROA	Return on assets
ROE	Return on equity
ROI	Return on investment
Quick ratio	Yrityksen lyhytaikainen maksuvalmius
DIP	Dependecy Inversion Principle
ISP	Interface Segregation Principle
LSP	Liskov Substitution Principle
OCP	Open-Closed Principle
SOLID	lyhenne viidelle suunnitteluperiaattelle
SRP	Single Responsibility Principle
API	Application Programming Interface
Botti	Itsenäinen tietokoneohjelma
CSV	Comma-separated values -tiedostomuoto
JSON	JavaScript Object Notation -tiedostomuoto
UML	Unified Modeling Language
URL	Uniform Resource Identifier, verkkosivun osoite
XML	Extensible Markup Language

1. JOHDANTO

Algoritminen kaupankäynti on tietokoneilla automatisoitua kaupankäyntiä esimerkiksi osakemarkkinoilla. Automatisoidussa kaupankäyntijärjestelmässä voidaan hyödyntää erilaisia sijoitusstrategioita sekä algoritmeja, joita tietokone noudattaa osto- ja myyntipäätöksien tekemisessä. Ihmisen ei tarvitse puuttua toimivan järjestelmän toimintaan, vaan tietokone tekee kaikki toimeksiannot ja kaupat. Ainoastaan algoritmien suunnittelu, kehittäminen ja järjestelmän ylläpito jäävät ihmisen tehtäväksi. Algoritmeilla ja tietokoneen nopealla reagoinnilla pyritään tekemään voittoa markkinoilla. Toimivalla algoritmilla voidaan päihittää ihminen ja markkinoiden indeksit. Lisäksi algoritmit ovat hyviä tilanteissa, joissa ihmiset olisivat liian hitaita tekemään päätöksiä ja toimeksiantoja markkinoille. Tällainen automaattinen järjestelmä soveltuu hyvin siis aktiiviseen kaupankäyntiin. Toki on myös mahdollista luoda algoritmi, joka etsii ja sijoittaa tuottaviin pitkän ajan sijoituskohteisiin.

Tämä kandidaatintyö vastaa kysymykseen: ”Miten suunnitellaan algoritmiseen kaupankäyntiin soveltuva järjestelmä tai ohjelma?” Järjestelmän tehtävä on mahdollistaa automaattinen kaupankäynti pörssissä sekä erilaisten kaupankäyntistrategioiden ja eri välittäjien alustojen hyödyntäminen.

Yksi algoritmisen kaupankäynnin muoto on nimeltään ultranopea kaupankäynti (engl. high-frequency trading). Siinä toimeksiantoja tehdään sekunnin murto-osissa ja kauppoja syntyy paljon. Tällaiselle järjestelmälle on tärkeää nopeat yhteydet ja mahdollisimman pieni latenssi välittäjän järjestelmiin ja markkinoille. Tässä kandidaatintyössä ei kuitenkaan perehdytä ultranopeaan kaupankäyntiin sen tarkemmin vaan enemmän muunlaisiin algoritmeihin ja järjestelmän suunnitteluun.

Aluksi luvussa kaksi perehdytetään lukija kaupankäynnin teoriaan ja erityisesti niihin asioihin, joita hyödynnetään tämän kaupankäyntijärjestelmän sekä algoritmien suunnittelussa. Seuraavaksi luvussa kolme kerrotaan yleisistä suunnitteluperiaatteista ohjelmistojen tekemisessä. Luvussa neljä yhdistetään aiempien lukujen teoriaa ja luodaan suunnitelma kaupankäyntijärjestelmän ohjelmointia varten. Lopuksi luvussa viisi käydään läpi tulokset ja luvussa kuusi yhteenveto koko kandidaatintyöstä.

2. KAUPANKÄYNNIN JA SIJOITTAMISEN TEORIAA

Tässä luvussa avataan kaupankäynnin ja sijoitustoiminnan teoriaa. Teoria käsittelee erityisesti erilaisia analyysikeinoja kaupankäyntiin ja teoriaa aktiiviseen kaupankäyntiin. Lisäksi luvussa kerrotaan myös erilaisista lisäkeinoista kaupankäyntiin kuten lyhyeksi myynnistä ja velalla sijoittamisesta.

2.1 Sijoitustoiminta

Kaupankäynti osakkeilla ja arvopapereilla tapahtuu pörsseissä, joita on maailmassa useita. Lähes jokaisella maalla on oma pörssinsä, jossa sen maan julkisten yritysten osakkeilla voidaan käydä kauppaa. Markkinoilla on olemassa paljon erilaisia sijoitustuotteita, joiden kautta voi sijoittaa johonkin yritykseen, materiaaliin tai valuuttaan [1]. Eri välittäjät tarjoavat erilaisia mahdollisuuksia sijoittamiseen, eikä kaikkien välittäjien kautta ole mahdollista käydä kauppaa kaikissa maailman eri pörsseissä.

Sijoittaminen osakkeisiin perustuu yritysten kasvuun ja tuottoon. Jotkin yritykset maksavat osinkoa eli palauttavat voittoja sijoittajille rahana. Tällaiseen yritykseen sijoittaminen voi olla kannattavaa pitkällä aikavälillä, vaikka osakkeen hinta ei juurikaan nousisi. Toisaalta on myös olemassa yrityksiä, joiden osakkeen hinta kasvaa nopeasti, eivätkä ne välttämättä maksa osinkoa ollenkaan. [1]

Yritysten osakkeiden arvo ja hinta eivät aina ole sama, ja tällöin hinnan laskiessa sen oikean arvon alle tulee kyseisestä osakkeesta kannattava sijoitus. Monesti nuorien yritysten arvo voi kasvaa nopeasti, vaikkei osakkeen hinta kasvaisikaan. Lopulta jossain vaiheessa huomataan osakkeen oikea arvo ja hinta tulee tällöin nousemaan nopeasti oikeaan arvoonsa. Tällaisia tilanteita tapahtuu markkinoilla useasti, jolloin jokin osake on hinnoiteltu huomattavasti korkeammaksi tai matalammaksi verrattuna sen oikeaan arvoon.

Myös osakekurssien ollessa laskussa on mahdollista tehdä voittoa. Tämä tapahtuu myymällä osakkeita korkeampaan hintaan, jotta voidaan ostaa sama määrä osakkeita takaisin halvemmalla [1]. Myydyt osakkeet on lainattu välittäjältä, ja jossain vaiheessa ne täytyy ostaa takaisin välittäjälle. Jos hinta meneekin korkeammaksi kuin myydessä, tulee

tappiota, mutta ostettaessa osakkeet uudelleen halvempaan hintaan saadaan aikaan voittoa. Tätä kutsutaan lyhyeksi myynniksi (englanniksi short selling) tai shorttaamiseksi. Lyhyeksi myynti riippuu välittäjästä eikä se ole aina mahdollista. Lisäksi siinä asetetaan yleensä oma osakesalkku pantiksi välittäjälle, jolloin mahdolliset tappiot saadaan korvattua välittäjälle, jos lyhyeksi myynti ei onnistukaan toivotulla tavalla. Tuottoa lyhyeksi myynnissä ei ole mahdollista saada moninkertaisena, mutta häviölle voi jäädä moninkertaisesti, jopa huomattavasti enemmän kuin on alun perin sijoittanut. Se on siis melko riskialtista sijoittamista, mikäli osakkeen hinta lähtee yllättävään nousuun. [2]

Monesti piensijoittajilla ei ole kovin suuri pääoma, joten monet välittäjät tarjoavat erityisesti aktiiviseen kaupankäyntiin hyödyllisen, mutta riskialttiin keinon kasvattaa tuottoja. Sijoittaja voi siis ottaa lainaa välittäjältä ja käyttää tätä rahaa sijoittamiseen. Mikäli tällä lainatulla rahalla hankittu positio alkaa mennä negatiiviselle puolelle voi välittäjä vaatia sijoittajalta lisää vakuuksia. Tässä tilanteessa sijoittaja voi myydä sijoituksensa tappiolla, lisätä tilille lisää rahaa tai jollain muulla tapaa antaa vakuus sijoituksesta välittäjälle. Tätä kutsutaan vakuuspyynnöksi (englanniksi *margin call*). [2]

2.2 Sijoitustuotteet

Sijoittamista varten on olemassa monenlaisia sijoitustuotteita. Osa tuotteista sisältää paljon riskiä ja mahdollisesti suuria tuottoja, mutta on myös olemassa paljon riskittömämpiäkin tuotteita, joiden tuotot ovat myös siten paljon maltillisempia.

Yksi yleisimmistä sijoituskohteista on yritysten osakkeet. Niiden kautta voi omistaa osan yrityksestä ja tehdä voittoa yrityksen kasvaessa. Lisäksi osakkeilla voi saada äänivaltaa yrityksen päätöksenteossa. Jotkin yritykset maksavat osinkoa omistajilleen, eli ne palauttavat osan voitoista osakkeiden omistajille. Tämä summa ei ole yleensä kuitenkaan kuin muutamia prosentteja tuotosta, mutta pitkällä aikavälillä osingot tuottavat huomattavan summan. Osakkeitakin voi olla erilaisia, kuten niin sanottuja etuoikeutettuja osakkeita ja tavallisempia kantaosakkeita [1]. Etuoikeutetut osakkeet tarjoavat omistajilleen etuoikeuden esimerkiksi osinkojen saamisessa. Yrityksen maksaessa osinkoa maksaa yritys mahdolliset osingot ensiksi etuoikeutetuista osakkeista ennen kantaosakkeille osingon maksamista. [2]

Rahastojen kautta voidaan sijoittaa useampaan osakkeeseen kerralla. Rahastoja on erilaisia, kuten tiettyihin sektoreihin tai tietyn maan osakkeisiin perustuvia, mutta perusidea on se, että rahastonhoitaja ostaa rahastoon haluamiansa osakkeita ja rahaston tuotto

perustuu näiden osakkeiden menestykseen. Lisäksi rahastoissa on yleensä vielä lisäkuluja, joilla katetaan kaupankäynnistä ja rahaston hallinnasta koituvat kulut rahastonhoitajalle. [1]

ETF-rahastot (*Exchange Traded Fund*) ovat pörssinoteerattuja rahastoja, eli niillä voidaan käydä kauppaa pörssissä samalla tavoin kuin osakkeilla. ETF:issä on yleensä huomattavasti pienemmät kulut kuin normaaleissa rahastoissa, ja siten ne voivat olla paljon tuottavampia pitkällä aikavälillä. Monet ETF-rahastot ovat indeksirahastoja, eli ne seuraavat tai jäljittelevät jotain tiettyä indeksiä ja antavat samanlaista tuottoa kuin henkilö ostaisi itse kaikkia indeksissä olevia osakkeita. [1]

Yksi alhaisen riskitason sijoituskeino on ostaa joukkovelkakirjoja. Ne ovat usealta taholta saadun lainan vakuudeksi annettuja velkakirjoja. Yleensä valtiot tai yhtiöt laskevat liikkeelle joukkovelkakirjoja, kun ne haluavat lainata rahaa. Joukkovelkakirjojen tarjoama tuotto on tasaista ja erääntyessä maksetaan lainattu pääoma takaisin sijoittajille [1]. Näillä voidaan käydä pörssissä kauppaa samalla tavoin kuin osakkeilla [2].

Sijoituskohteina voivat olla myös erilaiset hyödykkeet kuten maanviljelytuotteet, metallit, energia, rakennusmateriaalit ja jalokivet. Näistä esimerkiksi energiamarkkinoilla maakaasu ja öljy ovat aktiivisesti vaihdettua tavaraa. Metalleista vaihdetuimpia ovat kulta sekä muut teollisuudessa käytetyt materiaalit. [1]

Toimeksiantojen arvon perusteella valuuttamarkkina on maailman suurin markkina [2]. Valuuttaan sijoittaminen on houkuttelevaa alhaisten kulujen takia sekä siksi, että valuuttamarkkinat ovat viikonloppuja lukuun ottamatta auki kellon ympäri. Valuuttamarkkinat ovat kansainväliset, ja ne liittyvät aina kahden eri valtion välisten valuuttojen vaihtamiseen. Ulkomaanosakkeisiin sijoittaminen vaatii yleensä kyseisen maan valuuttaa osakkeiden ostamiseen. Tällöin sijoittaja joutuu vaihtamaan rahojaan toiseen valuuttaan ja tällöin samalla osallistuu valuuttamarkkinoille. [1]

Johdannaiset ovat eniten riskiä sisältävien pörssissä listattujen sijoitustuotteiden joukossa. Johdannaisten arvo perustuu jonkin kohde-etuuden kuten osakkeen arvoon, josta johdannaisen arvo on johdettu. Kohde-etuutena johdannaiselle voi toimia lähes mikä vain asia, mutta yleisimpiä kohde-etuuksia ovat indeksit, valuutat, hyödykkeet, osakkeet ja velkakirjat. Johdannaisista yleisimmät ovat optiot ja futuurit. Optio antaa oikeuden ostaa tai myydä kohde-etuus tiettynä hetkenä tiettyyn hintaan tulevaisuudessa [1]. Toisaalta futuuri on sopimus, jolla sitoudutaan ostamaan tai myymään kohde-etuus tiettynä

hetkenä tulevaisuudessa sopimushetkellä sovittuun hintaan [1]. Lisäksi on olemassa muita johdannaisia kuten warrantit, turbowarrantit, bull- ja bear-sertifikaatit sekä mini-futuurit [2]. [1]

2.3 Aktiivinen sijoittaminen

Monille helpoin ja kannattavin sijoituskeino on sijoittaa passiivisesti indekseihin rahastojen tai ETF:ien (Exchange Traded Fund) kautta. Tällöin siis sijoitetaan tasaisesti useisiin osakkeisin tai arvopapereihin eikä sijoittajan tarvitse seurata niin tarkasti, mitä markkinoilla tapahtuu. Aktiivisella sijoittamisella pyritäänkin saavuttamaan parempi tuotto kuin, mitä passiivisella sijoittamisella saataisiin esimerkiksi jonkin indeksin kautta. Pyritään siis poimimaan aina niitä osakkeita, jotka olisivat tuottavimmat sillä hetkellä.

Usein aktiiviset rahastot häviävät indeksille tai ovat valinneet samoja osakkeita kuin indeksissä, jolloin tuotto on lähellä indeksin tuottoa, mutta korkeammilla kaupankäyntikuluilla. Aktiivisessa sijoittamisessa on myös uhkana se, että useista toimeksiannoista johtuvat kaupankäyntikulut syövät tuottoa. Siksi olisi hyvä löytää välittäjä, joka tarjoaa mahdollisimman alhaiset kulut kaupankäyntiin. [3]

Aktiivinen sijoittamisen eräs muoto on päiväkauppa (day trading), jolloin päivän sisällä myydään ja ostetaan osakkeita voiton tekemiseen. Tällöin yleensä pyritään saamaan muutaman prosentin hinnanmuutoksista voittoja. Tällöin jopa senttien muutokset osakkeen hinnassa tuottavat voittoja, kun käytetään suuria osakemääriä kaupoissa. Päiväkaupassa on myös yleistä lyhyeksi myynti, jossa samalla tavoin voidaan saada senttien muutoksista huomattavia voittoja myymällä suuria osakemääriä kerralla. Lisäksi johdannaisten hyödyntäminen on yleistä aktiivisessa lyhyen aikavälin kaupankäynnissä, koska johdannaisten arvo liikkuu huomattavasti enemmän lyhyessä ajassa kuin varsinaisten kohde-etuuksien. [3]

Kauppaa voi käydä myös hieman pidemmällä aikavälillä. Esimerkiksi huomataan arvopaperin kehityksessä jonkinlainen lyhyen ajan trendi, joka kestää muutamista päivistä viikkoihin. Tällöin voidaan mennä omassa kaupankäynnissä trendin mukana ja ottaa hyöty tästä markkinaliikkeestä. Tätä kutsutaan swing treidaamiseksi (englanniksi swing trading). [3]

2.4 Fundamentaalinen analyysi

Fundamentaalinen analyysi on strategia sijoittamisessa, joka perustuu arvopaperien arvonmäärityksen perusasioihin. Nämä perusasiat sisältävät muun muassa yritysten raporttien, tuloslaskelmien ja taseiden tutkimista sekä niistä saatavien tunnuslukujen tulkitsemista arvopaperin arvon määrittämiseksi. Fundamentaalisessa analyysissä pyritään selvittämään yritysten taloudellinen tilanne, millainen asema yrityksellä tai sen tuotteilla on verrattuna sen kilpailijoihin sekä arvioida yrityksen kehitystä tulevaisuudessa. Näillä tiedoilla voidaan arvioida, onko yrityksen osakkeen markkinahinta oikea kyseisten lukujen pohjalta. Tällä tavoin yleensä etsitään kannattavia tai alihinnoiteltuja sijoituskohteita, joita voidaan ostaa ja omistaa useita vuosia. Vaikka fundamentaalinen analyysi on enemmän pitkäntähtäimen sijoittamista varten, se on hyvä apuväline kaupankäyntiin sijoituskohteiden etsimisessä ja sijoituspäätöksien vahvistamisessa teknisen analyysin rinnalla. [4]

Monet fundamentaalisen analyysin tunnusluvut perustuvat yrityksen taloudelliseen tilanteeseen. Tärkeitä tunnuslukuja ovat muun muassa yrityksen hinta voittoihin suhteutettuna (P/E -luku), hinta myyntiin suhteutettuna (P/S -luku), osinkokohtainen tulos (EPS), osinkojenmaksu suhdeluku, kate-%-luvut, oman pääoman tuotto (ROE), kokonaispääoman tuotto (ROA), sijoitetun pääoman tuotto (ROI), yrityksen maksuvalmiutta kuvaavat luvut quick ratio ja current ratio, liikevaihto, pääoman käyttö, yrityksen velan määrä ja velan suhde pääomaan. Tunnuslukuja on myös monia muita näiden lisäksi. [4]

2.5 Tekninen analyysi

Tekninen analyysi perustuu osakkeiden hintaan ja kurssien vaihteluun, joista pyritään etsimään ennustettavia säännönmukaisuuksia. Tärkein tehtävä teknisellä analyysillä on pyrkiä havaitsemaan erilaiset trendit ja kuviot osakekursseista sekä lopulta hyödyntää havaittua informaatiota sijoituspäätöksissä. Käytännössä tekninen analyysi on pörssikurssikaavioiden tulkitsemista erilaisten indikaattorien avulla. [5]

Teknisen analyysin hyödyntäminen perustuu siihen oletukseen, että osakkeiden hintoihin on jo otettu huomioon fundamentaalinen puoli ja muut taustalla olevat syyt. Osakkeiden on siis oltava hinnoiteltu suurin piirtein jo hintaan, missä niiden kuuluisi olla yrityksen tuloksen perusteella. Siten tekninen analyysi perustuu kokonaan osakkeen hinnan historiatietoihin eikä siinä huomioida tekeekö yritys tappiota vai voittoa. Siksi tekninen analyysi ei välttämättä yksinään ole kannattava keino kaupankäynnille vaan sitä kannattaa käyttää yhdessä muunlaisten analyysien kanssa. Tekninen analyysi voi olla esimerkiksi

hyödyllinen väline ostojen ja myyntien ajoittamiseen parhaimman tuloksen saamiseksi.[5]

Koska ihmiset ovat tunteellisia ja usein myös irrationaalisia, pörssikurssin kuviot muodostuvat yleensä ihmisten tunteiden johdattamana. Siten tekninen analyysi voidaan määritellä myös sijoittajien kollektiivisen psykologian tulkintana. Sijoittajan pahin vihollinen on hänen omat tunteensa. Usein siis pelko, toivo ja ahneus ohjaavat ihmiset tekemään kannattamattomia sijoituspäätöksiä. [5]

Teknisessä analyysissä käytettävät päätyökalut ovat kynttilänjalkagraafit ja indikaattorit. Kynttilänjalkakaaviossa arvopaperin hintaa esitetään eräänlaisina kynttilöinä. Jokainen kynttilä kuvastaa yhtä ajan yksikköä kuten päivä, tunti tai minuutti riippuen kaavion aikavälistä, jolla dataa esitetään. Kynttilän paksu osa kuvastaa hintaa, jonka verran arvopaperin hinta muuttui kyseisenä ajanjaksona. Ylä- tai alapuolella olevat kynttilänlangat kuvastavat hintoja, joissa arvopaperin hinta oli käynyt ajanjakson aikana, mutta ei ollut pysynyt sillä tasolla. Kynttilän väri on vihreä, jos sen aloitushinta on ollut pienempi kuin hinta ajanjakson päätyttyä eli osakkeen hinta on tällöin noussut. Ja päinvastoin hinnan laskiessa kynttilä on punainen. [3]

Indikaattorit voidaan jakaa kolmeen eri ryhmään sen mukaan mitä tietoa ne tarjoavat. Nämä ryhmät ovat trendejä seuraavat indikaattorit, oskillaattorit ja sekalaiset indikaattorit. Trendejä seuraavat indikaattorit ovat hyödyllisiä erilaisten trendien varmistamisessa ja toimivat parhaiten markkinoiden liikkuessa, mutta kurssien ollessa tasaista eivät nämä indikaattorit toimi enää yhtä hyvin. Oskillaattori-indikaattorit auttavat yleensä tunnistamaan käännöksiä kurssin kehityksessä. Sekalaisessa ryhmässä olevat indikaattorit antavat näkymää sijoittajien massapsykologiasta, eli siitä, kuinka sijoittajat käyttäytyvät ryhmänä. [3]

Indikaattoreita voidaan jakaa myös sen mukaan, kuinka ne esitetään kurssikaavion yhteydessä [6]. Overlay-tyyppisistä eli kurssikaavion päällä esitettävistä indikaattoreista yleisimpiä ovat liukuvat keskiarvot (MA, Moving Average) ja bollinger bands -indikaattori. Yleisimmät oskillaattori -tyyppisistä indikaattoreista ovat RSI ja MACD. Lisäksi on accumulator-tyyppisiä indikaattoreita, jotka perustuvat itsensä aiempiin arvoihin toisin kuin muun tyyppiset indikaattorit, jotka perustuvat osakkeen hinnan aiempiin arvoihin. Tällaiset accumulator -indikaattorit ovat usein kaupankäyntivolyymiin perustuvia. [6]

Mikään indikaattori ei toimi yksinään hyvin vaan täytyy valita useampi parempien myynti- tai ostopäätösten tekemistä varten. Kun useampi indikaattori indikoi samanlaista tapahtumaa, ovat signaalit vahvempia. Jokaisella indikaattorilla on yleensä omat arvot, jotka indikoivat tiettyä asiaa. Esimerkiksi RSI –indikaattorilla, joka vaihtelee arvojen 0 – 100 välillä, yli 70 kuvastaa sitä, että osake on yliostettu ja se voisi tarkoittaa hyvää paikkaa myydä osake pois ja alle 30 tarkoittaa, että osake on ylimyyty ja silloin voisi olla hyvä paikka ostaa osaketta. [5]

3. OHJELMISTOJEN SUUNNITTELUPERIAATTEET JA -MALLIT

Tässä luvussa käsitellään ohjelmointiin ja erityisesti ohjelmistojen suunnitteluun liittyvää teoriaa. Aluksi luvussa käydään läpi olio-ohjelmoinnin perusperiaatteita, jonka jälkeen käydään läpi olio-ohjelmointia hyödyntäviä suunnittelumalleja. Lisäksi luvussa kerrotaan ohjelmiston laadusta ja laatukriteereistä.

3.1 Olio-ohjelmoinnin periaatteet

Monet ohjelmistot ovat monimutkaisia, mutta niin on myös maailma tietokoneiden ulkopuolella. Olio-ohjelmointi perustuukin samanlaisiin elementteihin kuin oikea maailma. Esimerkkinä tähän voidaan ottaa vaikka kasvit. Kasvit koostuvat juurista, varresta ja lehdistä. Kaikilla näillä on oma tehtävänsä kasvin toiminnassa. Juuret imevät maasta vettä ja mineraaleja. Varsi on yhteydessä juuriin ja lehtiin sekä se välittää juurien keräämät aineet lehtiin. Lehdet vuorostaan käyttävät vettä ja mineraaleja muodostaakseen ravintoa kasville fotosynteesin avulla. Jokainen kasvin osa voidaan jakaa vielä pienempiin osiin, joilla jokaisella osalla on myös oma tehtävänsä. Kasvin kaikki osat hoitavat oman osuutensa ja luovat yhdessä monimutkaisen kokonaisuuden. Samalla lailla olio-ohjelmoinnissa ohjelmat koostuvat olioista tai luokista, joilla on omat tehtävänsä ja rajapintoja toistensa kanssa. [7]

Olio-ohjelmoinnissa yksi tärkeä käsite on modulaarisuus, joka tarkoittaa, että ohjelmat koostuvat useista moduuleista yhdessä. Jokaisella moduulilla on yleensä oma tehtävä ja moduulit voivat koostua useista luokista. Luokka on siis kokonaisuus, jolla on sisäinen toiminnallisuus ja rajapinta, jonka kautta luokan kanssa voi kommunikoida. Tämä sisäinen toiminnallisuus voidaan piilottaa luokan sisään ilman, että luokan kutsujan tarvitse tietää kuinka luokka on toteutettu [8]. Kutsujan tarvitsee tietää vain mitä toiminnallisuuksia luokan rajapinta tarjoaa. Tätä kutsutaan kapseloinniksi [8].

Luokka on pohja, jonka perusteella voidaan luoda olio tai useita olioita. Jokainen olio on rakennettu luokan pohjalta, mutta oliot voivat keskenään olla erilaisia sisältä. Ne sisältävät saman toiminnallisuuden, mutta niille annettu ja niiden sisältämä tieto voi olla erilaista, jolloin samalla luokalla saadaan useita rakennuspalikoita ohjelmaan. Esimerkiksi kaupankäyntijärjestelmää ohjelmoidessa voidaan luoda luokka osakkeelle, joka sisältää osakkeen nimen, hinnan ja muita tietoja. Kun luokka on määritelty koodissa, voidaan

luoda useita olioita luokan pohjalta. Jokainen olio sisältää siis saman rakenteen, mutta oliot voivat sisältää eri osakkeiden tietoja. [8]

Toinen tärkeä käsite olio-ohjelmoinnissa on periytyminen. Tämä käsite tarkoittaa sitä, että luokka voi periä toiselta luokalta ominaisuuksia ja lisäksi sisältää omia ominaisuuksia, joita alkuperäisellä luokalla ei ollut [8]. Periytymiseen liittyy vahvasti abstrakti luokka, joka määrittelee mitä joukko toisiinsa liittyviä luokkia voi tehdä. Abstrakti luokka on siis pohja tai määritelmä muille luokille, jotka käyttävät pohjanaan tätä abstraktia luokkaa ja perivät sen ominaisuudet itseensä [8]. Lisäksi abstraktin luokan käsitteeseen liittyy toinen tärkeä käsite nimeltä polymorfismi. Tämä tarkoittaa sitä, että voidaan viitata johdettuihin luokkiin samalla tavoin, mutta saadaan erilaista toiminnallisuutta sen luokan mukaisesti, johon viitataan [8]. Esimerkiksi voidaan määritellä abstrakti luokka arvopaperille, joka sisältää tiedot kuten arvopaperin nimen ja hinnan. Kun luodaan siis luokka osakkeelle, voi se saada pohjan abstraktista luokasta. Tällöin osakeluokkaan ei tarvitse erikseen määritellä attribuutteja “nimi” ja “hinta”, sillä osakeluokka perii nämä ominaisuudet abstraktista arvopaperiluokasta. Osakeluokka voi sisältää muita ominaisuuksia, joita pohjalla oleva kantaluokka ei sisällä tai se voi määritellä uudelleen kantaluokan metodeja. Esimerkiksi osakeluokalle voidaan nimen ja hinnan lisäksi määritellä osakkeen tyyppi, yrityksen kokonimi sekä yrityksen muita tietoja.

Olio-ohjelmointi helpottaa monimutkaisten ohjelmien rakentamista yhdistämällä luokkia, joilla jokaisella on omat tehtävänsä. Kaupankäyntijärjestelmää luodessa voidaan käyttää näitä olio-ohjelmoinnin periaatteita rakentamalla järjestelmä useista eri komponenteista. Järjestelmän komponentit hoitavat jokainen oman tehtävänsä ja yhdistettynä luovat toimivan kokonaisuuden.

3.2 Suunnitteluperiaatteet

Suunniteltaessa ohjelmistoa on tiedettävä vaatimukset ohjelmistolle ja mitä ominaisuuksia sillä täytyy olla. Tämän jälkeen voidaan suunnitella ohjelmiston rakenne. Ohjelmisto tulisi toteuttaa vasta suunnittelun jälkeen, ja lopulta ohjelman ollessa valmis se kannattaa testata virheiden varalta. Valmis ohjelmisto voidaan ottaa käyttöön. Käyttöönotetussa ohjelmistossa voi ilmetä uusia tarpeita ominaisuuksille tai ohjelman toiminnassa voi olla virheitä, jolloin sitä täytyy ylläpitää. Ohjelmiston elinkaari tapahtuu siis seuraavassa järjestyksessä: 1. vaatimukset, 2. suunnittelu, 3. toteutus, 4. testaus, 5. julkaisu ja 6. ylläpito. Hyvin suunniteltu ohjelmisto helpottaa sen toteuttamista, ylläpitämistä sekä muutosten tekemistä. [9]

Ohjelmistojen suunnittelun tärkeimpiä perusperiaatteita ovat modulaarisuus ja osittaminen [9]. Lisäksi ohjelmistoa suunnitellessa kannattaa toteutus pitää mahdollisimman yksinkertaisena ja suoraviivaisena. Myös abstraktioiden hyödyntäminen ja olio-ohjelmoinnin periaatteiden hyödyntäminen on hyödyllistä. Luvussa 3.1 kerrottiin ohjelman osittamisesta useisiin komponentteihin olio-ohjelmoinnissa. Ohjelmisto ositettaessa saadaan oikeasti luotua toimiva monimutkainen järjestelmä, joka olisi hyvin vaikeaa luoda yhtenä suurena kokonaisuutena. Suurin osa tosielämän ohjelmistoista ovat suuria, miljoonien koodirivien ohjelmistoja. Laajamittaiset ohjelmistot onkin yleensä jaettu pienemmiksi osiksi ja jokainen osa pystyy toimimaan itsenäisesti hoitaen oman tehtävänsä. Jos halutaan siis luoda monimutkainen ja moderni tosielämän käyttöön soveltuva ohjelmisto, täytyy ohjelmiston suunnittelun sekä sen implementaation molempien olla korkealaatuisia. Hyvä suunnittelu helpottaa huomattavasti ohjelmiston kehittämistä sekä sen ylläpitämistä. [10]

Korkealaatuisen ohjelmiston suunnittelussa kannattaakin käyttää hyödyksi suunnittelumalleja sekä noudattaa olio-ohjelmointiin perustuvia SOLID-periaatteita. SOLID lyhenne tulee viidestä eri suunnitteluperiaatteesta: Single responsibility principle (SRP), Open-closed principle (OCP), Liskov substitution principle (LSP), Interface segregation principle (ISP) ja Dependency inversion principle (DIP). SRP:n mukaan luokalla tulisi olla vain yksi vastuualue. Ei luoda siis luokkaa, joka pystyy kaikkeen. Sen sijaan jaetaan ohjelma luokkiin, joilla kullakin on oma tehtävänsä. OCP:n mukaan ohjelmiston moduulien tai luokkien tulisi olla avoimia laajennuksille, mutta suljettuja muutoksille. Moduuliin voisi asettaa uusia erilaisia toiminnallisuuksia, mutta se pitäisi pystyä tekemään ilman, että moduulin olemassa olevaa toiminnallisuutta tai koodia muokataan. LSP on periaate, jonka mukaan ohjelmassa olevat oliot pystytään korvaamaan niiden alatyyppien tai -luokkien ilmentymillä ilman, että ohjelman korrektisuus muuttuu. ISP:n mukaan moduulien isommat rajapinnat tulisi jakaa pienemmiksi roolien mukaan. Tällöin tietyt rajapinnat toimivat tietyillä tarkoitusperillä sen sijaan, että ne olisivat yleiskäyttöisiä kaikissa tilanteissa. DIP:n mukaan moduulien riippuvaisuudet tulisi tarjota rajapintojen kautta muille luokille. Riippuvuudet ovat kätketty luokan toteutukseen, jolloin ne saadaan käyttöön helposti luomalla olio luokasta. [10]

3.3 Ohjelmiston laatu

Erityisesti sairaaloissa, pankeissa tai muissa yhteiskunnalle kriittisissä systeemeissä käytettävien ohjelmistojen on oltava laadultaan hyviä, jotta toiminta pysyy yllä virheettömänä ja ongelmattomana. Ominaisuudet kuten oikeellisuus, luotettavuus, eheys ja tehokkuus ovat todella tärkeitä tällaisissa systeemeissä. Myös muunneltavuus on erityisesti tärkeä ominaisuus ohjelmiston jatkokehityksen kannalta. [9]

Ohjelmiston laatua ja toimintaa voidaan määritellä muun muassa seuraavilla ominaisuuksilla: oikeellisuus, luotettavuus, tehokkuus, eheys, käytettävyys, huollettavuus, testattavuus, joustavuus, siirrettävyys, uudelleenkäytettävyys ja yhteentoimivuus. Oikeellisuus määrittää toimiiko ohjelmisto ja koodi sille määritetyllä tavalla ja toteuttaako ohjelma sille määritetyt vaatimukset. Luotettavuus voidaan määritellä todennäköisyytenä sille, suorittaako ohjelma määritellyn toiminnallisuuden oikeassa ympäristössä tietyssä ajassa. Tehokkuus puolestaan viittaa ohjelman reagointinopeuteen, eli siihen kuinka nopeasti ohjelma reagoi käyttäjän syötteeseen. Tehokkuus voidaan määritellä myös suoritettujen tapahtumien määränä tietyssä ajassa. Eheys kuvaa sitä, kuinka turvallisesti ohjelma käsittelee tietoja. Käytettävyys on ominaisuus, joka määrittelee voiko ohjelmaa ajaa helposti, ja kuinka helppoa käyttäjän on käyttää ohjelmistoa. Huollettavuus kuvaa sitä kuinka helppoa ohjelmiston bugeja on korjata ja kuinka helppoa ohjelmistoon on lisätä uutta toiminnallisuutta. Testattavuus kuvastaa ohjelmiston testaamisen helppoutta eli sitä, kuinka hyvin pystytään testaamaan koko ohjelmisto. Joustavuus kuvastaa ohjelmiston toimivuutta erilaisissa tapauksissa ja sitä, kuinka hyvin sitä voidaan muuttaa. Siirrettävyys on ominaisuus, joka kuvaa ohjelmiston toimivuutta eri laitteissa, käyttöympäristöissä ja käyttöjärjestelmissä. Uudelleenkäytettävyys helpottaa ohjelmiston tai sen osan käyttämistä uudelleen uudessa eri ohjelmistossa. Yhteentoimivuus on ominaisuus, joka määrittää kuinka hyvin ohjelma toimii muiden laitteiden tai ohjelmistojen kanssa. [9]

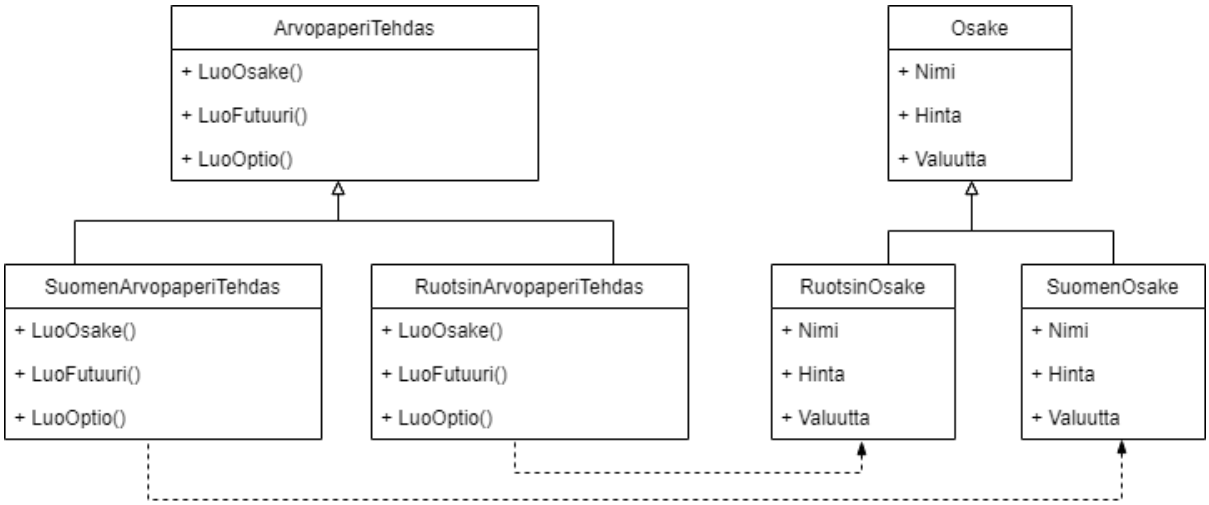
Kaikki ominaisuudet muodostavat yhdessä ohjelmiston laadun. Siksi ohjelmistot kannattaa suunnitella pitäen mielessä kaikki nämä ominaisuudet, jotta ohjelmiston laatu olisi mahdollisimman hyvä. Ohjelmistot voivat käyttötarkoituksesta riippuen suosia tiettyjen ominaisuuksien laadukkuutta ja tärkeyttä enemmän kuin toiset ohjelmistot. [9]

3.4 Ohjelmistojen suunnittelumallit

Ohjelmistojen suunnittelussa voidaan helpottaa ohjelmiston monimutkaisuutta ja kohottaa ohjelmiston laatua käyttämällä hyödyksi suunnittelumalleja. Suunnittelumalleissa tu-

lee hyötykäyttöön erityisesti olio-ohjelmoinnista periytyminen, jonka avulla saadaan uudelleenkäytettävyyttä koodille. Jokainen suunnittelumalli ratkaisee yleensä jonkin ongelman, mutta suunnittelumallit eivät aina sovellu kaikkiin tilanteisiin. Tällaisissa tilanteessa täytyy valita paras malli kuhunkin ongelmaan. Suunnittelumallit voidaan luokitella yleensä käyttötarkoituksensa perusteella joko luonti-, rakenne- tai käyttäytymismalleihin. Luontimallit ovat hyödyllisiä erilaisissa objektien luomiskäyttötarkoituksissa. Rakennemallit puolestaan soveltuvat luokkien tai olioiden rakenteiden määrittämiseen ja käsittelyyn. Käyttäytymismallit määrittävät tapoja, joilla luokat kommunikoivat keskenään ja kuinka vastuu jakautuu luokissa. [11]

Abstrakti tehdas on luontityypin suunnittelumalli, jonka tarkoitus on tarjota rajapinta olioperheiden tai toisistaan riippuvien olioiden luomiseen ilman, että määritellään niiden konkreettisia luokkia. Tämä mahdollistaa myös samanlaisen rajapinnan kaikille saman perheen olioille. Kuvassa yksi on esimerkki abstraktista tehtaasta. [11]



Kuva 1. Abstrakti tehdas luokan rakenne. Perustuu lähteeseen [11].

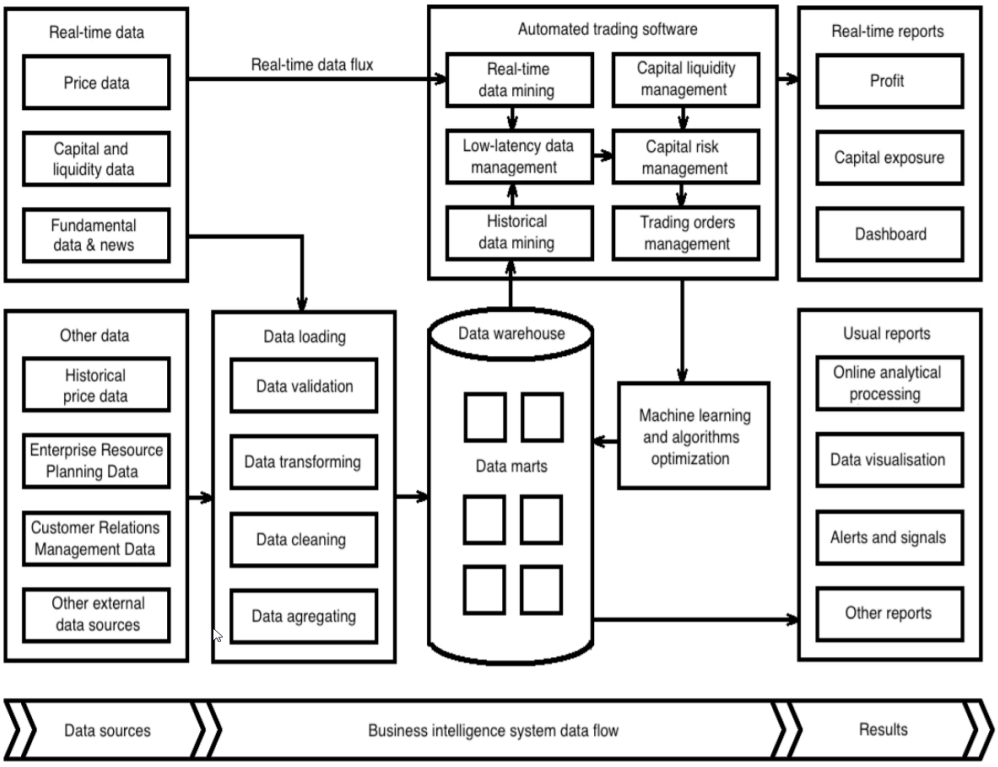
Abstraktilla tehtaalla voidaan siis luoda aliluokka tehtaita, joilla on oma rakenne, mutta julkinen rajapinta on sama. Tämä suunnittelumalli on hyödyllinen, kun täytyy valita yksi useista eri olioperheistä käyttöön tai olioperheet ovat suunniteltu toimimaan yhdessä. [11]

4. KAUPANKÄYNTIJÄRJESTELMÄN SUUNNITTELU

Tässä luvussa yhdistetään aiempien lukujen teoriaa ja suunnitellaan algoritmiseen kaupankäyntiin soveltuva automaattinen ohjelma eli kaupankäyntibotti. Suunnitelmassa pyritään ottamaan huomioon kaikki tärkeät asiat tällaisen kaupankäyntijärjestelmän kannalta. Varsinaista koodia ei tässä käydä läpi, mutta suunnitelman pohjalta pitäisi pystyä ohjelmoimaan toimiva järjestelmä. Luvun alkupuolella esitetään ohjelman kokonaisrakenne ja alaluvuissa käsitellään kutakin komponenttia tarkemmin. Luvun lopussa pohditaan tämän ohjelman toteuttamiseen soveltuvia ohjelmointikieliä.

4.1 Järjestelmän komponentit

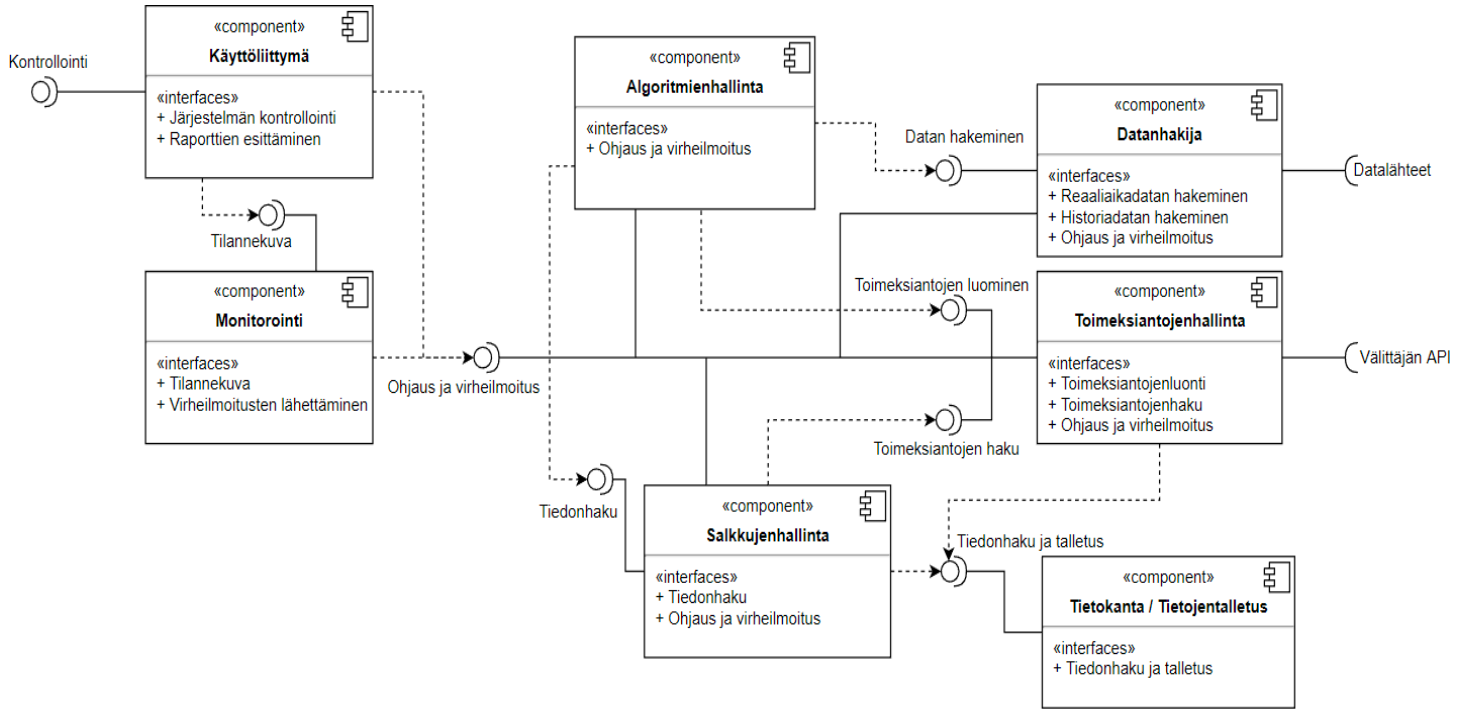
Automaattinen kaupankäyntiohjelma voi rakentua esimerkiksi kuvan 2 esittämällä tavalla. Ohjelmaan syötetään reaaliaikaista ja historiadataa pörssistä. Tästä datasta lasketaan tuottavia sijoituskohteita tai sijoitusajankohtia jonkin algoritmin mukaan. Saadulla tiedolla luodaan toimeksiantoja riskinhallinnan tarkistamana ja käydään kauppaa pörssissä. Järjestelmästä saadaan ulos tietona tuotto sekä muita raportteja ohjelman toiminnasta. [12]



Kuva 2. Tietovirta automatisoidun kaupankäyntijärjestelmän toiminnassa [12].

Aiemman perusteella voidaan määritellä eräänlaiset vaatimukset automaattiselle ja tuottavalle kaupankäyntijärjestelmälle. Näitä vaatimuksia ovat tuottoa tekevä algoritmi, riskin- sekä pääomanhallintasysteemit, toimivat rajapinnat välittäjien palveluiden kanssa ja datalähteet historiadatalle sekä reaaliaikaiselle pörssidatalle. Kaikki tämä pitäisi tietenkin myös automatisoida. Lisäksi olisi hyvä, että ohjelma tukee monia eri algoritmeja ja niitä olisi helppo lisätä myöhemmin järjestelmään.

Seuraavassa kuvan 3 UML-kaaviossa (Unified Modeling Language) esitetään tähän kandidaatintyöhön suunnitellun kaupankäyntijärjestelmän komponentit. Jokainen laatikko kuvastaa yhtä komponenttia ja katkoviivojen päässä olevat nuolet osoittavat komponentteihin, joihin kyseinen komponentti yhdistyy. Komponenttikaaviossa viivoilla yhdistyvät ympyrät kuvaavat kyseisen komponentin tarjoamia rajapintoja, ja puoliympyrät ilman palloa kuvaavat rajapintoja, joista komponentti itse on riippuvainen. [8]



Kuva 3. Suunnitelma kaupankäyntijärjestelmän komponenteista.

Jokainen kuvan 3 komponenteista koostuu useammista luokista. Näiden komponenttien toimintaa avataan tarkemmin seuraavissa alaluvuissa, mutta komponenttikaaviosta voidaan päätellä, kuinka järjestelmä toimii ilman, että tiedetään moduulien yksityiskohtaisempaa toteutusta.

4.2 Kaupankäyntialgoritmit ja -strategiat

Erilaisia kaupankäyntistrategioita on olemassa monenlaisia, eikä moni varmasti jaa tuottavimpiaan strategioitaan ilmaiseksi muille, joten tässä alaluvussa käydään läpi vain yksinkertaisia esimerkkejä algoritmeista ja niiden lisäämisestä tähän järjestelmään.

Algoritmikaupankäynnissä hyödynnetään koneoppimista, markkinauutisia, teknistä ja fundamentaalista analyysiä sekä muita keinoja tuottavien sijoitusten löytämiseksi. Niitä voidaan käyttää erikseen tai yhdistelemällä eri tavoin uuden tuottavan algoritmin luomiseksi. Yleisiä kaupankäyntistrategioita ovat trendien seuraaminen, osakkeen kiihtyvät hintakehitykset johonkin suuntaan, trendien kääntyminen, kuvioiden löytäminen osakkeen kurssista ja pörssiuutisiin perustuvat strategiat [13]. Näiden kaupankäyntialgoritmien kehittäminen vaatii strategian rakentamista, testaamista, optimointia ja ylläpitoa [6]. Tärkeitä asioita kaupankäyntialgoritmien kehittämisessä ovat testaaminen historiadatalla (eng. backtesting), strategian optimointi, suorituskyvyn mittaaminen ja parametrien päivittäminen [6]. On myös tärkeää ottaa huomioon kaupankäyntikulut algoritmia suunnitellessa ja testatessa, sillä jos kuluja ei huomioida voi muuten toimiva algoritmi muuttua kulujen kanssa tappiolliseksi [14]. Kuluja ovat muun muassa välittäjän palkkiot, voitoista aiheutuvat verot sekä tarjous- ja pyyntötasojen välisestä raosta (engl. bid-ask spread) aiheutuvat kulut [14]. Tarjous ja pyyntötasojen välisten erojen luoma kulu tarkoittaa sitä, jos sijoittaja ostaa osaketta tiettyyn hintaan sekä haluaa myydä samaan aikaan samaa osaketta niin yleensä osakkeen myynti- ja ostohinnan välissä voi olla useiden senttien väli [15].

Kaupankäyntiohjelmaan suunnitellussa algoritmimoduulissa hyödynnetään abstraktia luokkaa ja periytymistä. Algoritmeja voi olla useita erilaisia ja ne voivat toimia eri tavoin, mutta jos kaikki algoritmit rakennetaan saman rajapinnan päälle, voidaan lisätä helposti uusia algoritmeja järjestelmään ilman, että muutetaan olemassa olevaa järjestelmää. Kaikki algoritmit sisältävät siis saman rajapinnan, joka hoitaa pääasiassa toimeksiantosignaalien luonnin ja sisään syötetyn pörssidatan analysointia. Se kuinka signaalit muodostuvat jää itse algoritmin vastuulle, joka sijaitsee aliluokan toteutuksen sisässä.

Abstraktin algoritmikomponentin julkinen rajapinta koostuu pörssidatan syöttämisestä algoritmille sekä toimeksiantosignaalien lähettämismetodeista. Algoritmi itsessään voidaan toteuttaa sisältä muuten lähes millä tavalla vain. Strategia voi esimerkiksi koostua datan perusteella kannattavia osakkeiden valinnasta ja luoda niille ostosignaaleja tai käydä vain perinteisesti kauppaa muutamilla esivalituilla osakkeilla esimerkiksi teknisten

indikaattorien perusteella. Lisäksi järjestelmään voidaan kehittää backtestaus (backtesting) työkalu, joka osaa simuloida algoritmin toimintaa historiadatalla [16].

Yksi melko yksinkertainen tapa luoda kaupankäyntialgoritmi on käyttää teknisen analyysin indikaattoreita. Esimerkiksi osakkeen hintaan muodostuvia trendejä seuraava strategia voi käyttää hyödyksi teknisiä indikaattoreita kuten liukuvia keskiarvoja ja MACD -indikaattoria. Näillä indikaattoreilla voidaan luoda sääntöjä kuten “kun osakkeen hinta ylittää liukuvan keskiarvon yläpuolelle, ostetaan, ja kun hinta alittaa liukuvan keskiarvon alapuolelle myydään lyhyeksi”. Liukuvista keskiarvoista kuitenkin valita itse esimerkiksi 50 päivän tai 20 päivän liukuva keskiarvo ja kokeilla millä osakkeella tämä strategia voisi toimia. MACD indikaattoria voidaan käyttää esimerkiksi luomalla sääntö: “ostetaan kun MACD arvo on positiivinen ja myydään lyhyeksi, kun MACD arvo on negatiivinen”. Indikaattoreita voidaan myös yhdistellä luotettavamman strategian luomiseksi, jolloin saadaan virhesignaalien määrää vähennettyä. [6]

4.3 Käyttäjän ja salkun hallinta

Jotta tämä koko järjestelmä voisi toimia, täytyy pitää kirjaa kokonaisvarallisuudesta, omistetuista arvopapereista ja positioista. Tämä pitää luoda hyvin skaalautuvaksi, jos halutaan käydä kauppaa useiden välittäjien kautta. Kuten aiemmissa moduuleissa, myös tässä moduulissa hyödynnetään periytymistä ja luodaan abstrakti luokka salkulle. Jokaiselle välittäjän salkulle luodaan oma aliluokka ja lisäksi on salkkujenhallitsijaluokka, joka osaa yhdistää toimeksiannot ja muut tiedot oikeisiin salkkuluokkiin. Luokkien tiedot täytyy tietenkin tallettaa myös pysyvään muistiin esimerkiksi johonkin tiedostoon. Tällä estetään tietojen katoaminen ohjelman tai tietokoneen kaatuessa. Toinen keino toteuttaa useiden salkkujen toiminnallisuus olisi tallettaa kaikki tiedot tietokantaan ja käyttää vain yhtä hallitsijaluokkaa, joka osaa yhdistää tiedot oikein ja hakea tarvittavia tietoja tietokannasta.

Välittäjien palveluissa kaikilla käyttäjillä on olemassa yleensä seuraavat tiedot: käyttäjän ID, oletusvaluutta, kokonaissaldo, vapaana oleva saldo ja käytössä oleva vipuvoima [17]. Myös muita tietoja voi olla saatavilla kuten toteutunut voitto, realisoimaton voitto, käytetty lainamarginaali, avoimet kaupat ja avoimet toimeksiannot [17]. Salkkuluokkien tai tietokannan täytyy sisältää siis edellä mainitut tiedot, listan jokaisesta omistetusta arvopaperista ja arvopaperien tiedot kuten arvopaperin nimi, määrä, ostohinta ja ostohintojen keskiarvo. Salkkujen hallitsijaluokka on rajapintana kaikille eri salkuille ja osaa yhdistää tiedot oikeaan paikkaan.

Mikäli kauppaa käydään eri maan valuutalla kuin salkun oletusvaluutalla, täytyy ottaa huomioon valuuttakurssit. Esimerkiksi kun lasketaan salkun kokonaisarvo, täytyy muuttaa osakkeiden hinta sen maan valuutasta oletus valuuttaan ennen kuin tiedetään todellinen arvo. Järjestelmään tulisi siis tallettaa myös käytettyjä valuuttapareja kunkin osakkeen kohdalla ja valuuttakurssit kyseisenä ajankohtana. [17]

4.4 Kommunikaatio ulkoisten rajapintojen kanssa

Järjestelmässä on ulkoisia rajapintoja ainakin välittäjän järjestelmän sekä data- ja uutislähteiden kanssa, joista markkinadataa sekä muuta hyödyllistä dataa voidaan saada. Myös joidenkin välittäjien kautta voidaan saada kaupankäynnin kohteena olevan arvopaperin hinta- ja historiatiedot, mutta on hyvä olla muitakin lähteitä varalla, jos tiedon saanti katkeaa jostain syystä yhdestä lähteestä. Nämä tiedot yhdistetään järjestelmän sisäisen tiedon kanssa, jota on muun muassa avoimet positiot, salkussa olevat osakkeet, niiden määrä ja jäljellä oleva vapaa saldo.

Historiadataa voidaan yleensä hakea erilaisista verkosta löytyvistä tietolähteistä, jonka rajapintaan lähetetään pyyntö ainakin arvopaperin koodista, aikaväli sekä millä tarkkuudella data halutaan. Tarkkuus voi olla esimerkiksi päivän, tunnin, viiden minuutin tai vaikka minuutin tarkkuudella. Myös muita parametrejä voidaan asettaa. Esimerkiksi Yahoo! Finance API:iin pyyntö tehdään HTML protokollalla seuraavasti:

API URL = Perus URL + ? + “*parametri1*” + & + “*parametri2*” + & + … + & + “*parametri n* ” [6].

Siinä osoitteen alussa on URL osoite ja tämän jälkeen parametreina seuraa osakkeen koodi, datan aikaväli ja muut tiedot. Data voidaan datalähteen rajapinnasta riippuen saada esimerkiksi XML, JSON tai vaikka CSV-tiedoston muodossa. Haettu data täytyy tämän jälkeen lukea ja jäsentää ohjelman käytettäväksi sopivaan muotoon. [6]

Datanhakumoduuli voidaan toteuttaa perinnän avulla, jotta rajapinta olisi samanlainen omalle järjestelmälle riippumatta siitä, mistä lähteestä data tulee. Luodaan siis abstrakti luokka, jonka rajapinnassa olisi metodit datapyynnön lähettämiseen, datanparsinta ja datanpalautus sopivassa muodossa ohjelman käyttöön. Jokaisen datalähteen sisäinen toteutus voidaan piilottaa luokan sisään, ja koska kaikki datanhakuluokat periytyvät sa-

masta luokasta voidaan käyttää mitä vain tietolähdettä ilman erillistä integrointia järjestelmään. Tässä moduulissa voidaan käyttää luvussa 3.4 esiteltyä abstrakti tehdas -suunnittelumallia. Luodaan datayhteyksille abstrakti tehdas luokka, josta periytyy tehdasluokkia eri datalähteisiin. Tehdasluokat luovat uuden datayhteysluokan jokaiselle kyseiselle datalähteelle luodulle yhteydelle. Esimerkiksi datalähde A:sta halutaan hakea tietoa osakkeesta A ja osakkeesta B. Tällöin datalähde A:lle luotu tehdasluokka luo datayhteysluokat A:lle ja B:lle, joissa haetaan kyseisen osakkeen dataa. Tämä on hyödyllistä, kun tarvitaan useiden eri osakkeiden reaaliaikatietoja yhtäaikaisesti esimerkiksi usealle eri algoritmille.

Reaaliaikadatalle täytyy luoda omanlainen luokka, jossa dataa striimataan ja haetaan jatkuvasti markkinoiden ollessa auki [17]. Algoritmit tarvitsevat ainakin yhden tällaisen lähteen, jos aiotaan käydä kauppaa reaaliajassa. Tämän moduulin rajapinta koostuu siis datayhteyden luonti metodista, jossa määritetään haluttu kaupankäyntiväline. Lisäksi moduulissa on tietenkin rajapinta ohjelman sisälle päin, josta tietoa voidaan siirtää eteenpäin algoritmeille. Moduulista saadaan vähintään tieto sen hetkisistä pyyntö- ja tarjoustasoista arvopaperin hinnassa ja tämän tiedon aikaleima. Parametrina yhteydelle voidaan määrittää myös algoritmin tarvitsema aikaväli eli kuinka tiheään algoritmi tarvitsee uuden tiedon. Kaikki algoritmit eivät välttämättä tarvitse aivan jokaisen millisekunnin hintatietoa vaan esimerkiksi jokaisen minuutin tai puolen minuutin välein. [17]

Toinen tärkeä rajapinta on välittäjien rajapinnat, joiden kautta itse kaupankäynti toteutetaan. Sinne lähetetään toimeksiantopyyntöjä ja saadaan vastauksena toimeksiannon toteutumisen tilanne. Välittäjältä voidaan kysyä myös jonkin toteutumattoman markkinalla olevan toimeksiannon tilannetta. Tässä luokassa hyödynnetään tapahtumapohjaista arkkitehtuuria, jossa toiminta perustuu tapahtumiin. Kun systeemi saa ilmoituksen uudesta tapahtumasta reagoidaan tapahtumiin kullekin tapahtumalle määritetyllä tavalla. Myös tässä komponentissa hyödynnetään abstraktia luokkaa, jotta saadaan järjestelmä tarvittaessa toimimaan useiden eri välittäjien rajapinnoissa. [18]

Tieto toimeksiannosta välitetään toimeksiantoja luovalle moduulille. Moduulissa käytetään message broker -arkkitehtuurimallia viestien lähettämiseen ja vastaanottamiseen välittäjän rajapinnan kanssa. Järjestelmän toiminta ei saa pysähtyä siksi aikaa, kun odotetaan vastausviestiä välittäjän systeemistä. Tällainen message broker -arkkitehtuuri sisältää siis jonon lähtevistä viesteistä ja jonon saapuneista viesteistä. Se pystyy käsittelemään jonoon tulleita viestejä kunkin omalla vuorollaan ilman, että viestiliikenne tai järjestelmän toiminta katkeaa, kun odotetaan vastausviestiä tai kun viestiä käsitellään. [18]

4.5 Riskinhallinta

Riskinhallinta on tärkeä osa kaikessa kaupankäynnissä. Sillä rajoitetaan mahdollisia tappioita ja siten pidetään voitot mahdollisimman suurina. Tähän kaupankäyntijärjestelmään on hyvä lisätä myös jonkinlaista riskinhallintaa, joten tässä alaluvussa käsitellään mahdollisia riskinhallinta asetuksia ja järjestelmiä, joita ohjelmalle voidaan lisätä.

Salkkujenhallintaluokalle voidaan luoda riskinhallintajärjestelmä, joka laskee automaattisesti kaikkien salkkujen kokonaistuoton yhteensä ja seuraa kunkin algoritmin tuottoa reaaliajassa. Algoritmin tuoton perusteella voidaan säätää kyseiselle algoritmille myönnettyä pääomaa, jos algoritmi alkaa suorittamaan huonosti verrattuna muihin algoritmeihin. [13]

Riskinhallintaa voidaan lisätä myös toimeksiantojenhallinta moduulille, jossa luodaan toimeksiantoja markkinoille. Riskinhallinta voi yksinkertaisimmillaan olla stop-loss -tasojen eli keskeytystasojen asettamista kaikille toimeksiannoille. Stop-loss -tasot ovat toimeksiannoille määritettyjä hintatasoja, joissa osake myydään pois, mikäli sen hinta tippuu kyseiselle tasolle. Näin voidaan vähentää tappioita, jos osakkeen hinta lähtee yhtäkkiä syöksymään alaspäin. Stop-loss -taso voi olla tilanteesta ja riskinottohalukkuudesta riippuen esimerkiksi muutaman prosentin ostohinnan alapuolella. Toinen vaihtoehto, jota voidaan hyödyntää, on trailing stop eli osakkeen hinnan mukana kulkeva stop-loss taso. Tällöin stop-loss taso kulkee ylöspäin osakkeen hinnan mukana, ja kun hinta saavuttaa huippunsa ja hinta alenee hieman huipusta, pysäytetään kauppa ja myydään osake pois. Tässä tapauksessa saadaan pidettyä osa voitoista pysäyksen jälkeen sen sijaan, että pysäytettäisiin hinnan ollessa tappiolla. [16]

Riskinhallintaa on myös se, että panostetaan kauppoihin vain tietty osa kokonaissaldosta. Tämä voidaan myös asettaa toimeksiantojenhallintamoduuliin. Toimeksiantoja luodessa tarkistetaan vapaana oleva saldo ja salkun kokonaissaldo, jonka jälkeen lasketaan maksimisumma kyseiselle kaupalle. Maksimisumma lasketaan järjestelmälle tai algoritmille asetetun riskitason mukaisesti. Riskitaso voidaan määrittää esimerkiksi prosenttiluvulla kokonaissaldosta, joka ollaan valmiita häviämään yhdessä kaupassa. [12]

4.6 Osto- ja myyntisignaalit

Kun dataa on haettu, voidaan se syöttää algoritmeihin. Algoritmeissa muodostetaan kaupankäyntisignaaleja ja valitaan osakkeita syötetyn markkinatiedon sekä sisäisen salkkutiedon perusteella. Kaupankäyntisignaalit koostuvat ainakin toimeksiannon tyypistä eli

myydäänkö vai ostetaanko, vaihdettavan arvopaperin koodista sekä hinnasta, jolla toimeksianto toteutetaan. Volyymiä eli vaihdettavien arvopaperien määrää ei voida määritellä vielä, vaan se saadaan toimeksiantojenhallintamoduulista, joka suorittaa riskinhallintaa. Toimeksiantojenhallinnan riskinhallintajärjestelmä tarkistaa nykyisen tilanteen pääoman ja osakkeiden suhteen sekä asettaa sopivan volyymin kaupankäyntisignaalille halutun riskitason mukaisesti. Lisäksi riskiä voidaan vähentää ja kontrolloida asettamalla sopiva keskeytystaso (stop-loss) tälle toimeksiannolle. Lopuksi valmis toimeksianto lähetetään välittäjän järjestelmään ja odotetaan vastausta, onnistuiko toimeksianto, tai mikä sen tilanne on. [12]

Vastausviestin perusteella talletetaan tieto tietokantaan onnistuneista ja avoimista kaupoista. Talletettavia tietoja ovat ainakin välittäjän ottaman palkkion suuruus, toimeksiannontyyppi, osakkeiden määrä ja niiden hinta. Toimeksiantojenhallintamoduulin rajapintoihin kuuluu siis toimeksiantojen luominen, olemassa olevien toimeksiantojen hakeminen ja niiden tilanteen tarkistaminen välittäjältä sekä se pystyy tallettamaan tietoja toimeksiannoista tietokantaan.

4.7 Järjestelmän monitorointi ja virhetilanteisiin varautuminen

Kaupankäyntijärjestelmän suurin ongelmien aiheuttaja on kommunikaatioyhteyksien katkeaminen [13]. Kun pörssitietojen vastaanottaminen keskeytyy, ei toimeksiantoja luova algoritmi pysty reagoimaan markkinoiden liikkeisiin, ja tällöin saatetaan menettää hyviä tilaisuuksia, tai tehdä tappiota jo hankituilla osakkeilla. Tätä varten täytyy olla riskinhallintaa kuten keskeytystasoja (stop-loss) ja varatietolähteitä, jotka voidaan ottaa nopeasti käyttöön täyttämään tiedonhakuun syntynyt aukko. [13]

Myös muunlaisia virheitä voi sattua. Tällaisia ovat esimerkiksi välittäjän järjestelmän hajoaminen tai laiterikko laitteelle, jossa automaattinen kaupankäyntiohjelma pyörii. Näiden tapauksien varalta kaikki tieto tulisi olla myös kovalevyllä talletettuna esimerkiksi tietokannassa tai tiedostoissa. Toiminnan jatkuessa mitään tietoa ei ole tällöin kadonnut. Tietoja ei saa tallettaa siten pelkästään tietokoneen keskusmuistiin, koska keskusmuisti tyhjenee virran katketessa koneesta. [13]

Lokitiedostojen luonti on tärkeä osa virhetilanteista palautumisen ja virheiden selvittämisen kannalta. Kaikista ohjelman toiminnan kohdista olisi hyvä tallettaa jotain lokitiedostoon, jotta voidaan jäljittää virhetilanne sellaisen sattuessa. [13]

Yksi virheitä ennaltaehkäisevä osa kaupankäyntijärjestelmässä on heartbeating eli sydämensyke. Pitkäaikaisien tai pysyvien yhteyksien kunto datalähteisiin täytyy vähän väliä tarkistaa. Tämä tapahtuu lähettämällä sydämensyke datalähteestä järjestelmälle, jonka perusteella kaupankäyntiohjelma tietää yhteyksien olevan kunnossa. Jos tämä syke on epäsäännöllinen tai ylittää ennalta asetetun odotusajan, yritetään yhdistää uudelleen datalähteeseen. Mikäli esimerkiksi reaaliaikaisesta datalähteestä kadotetaan yhteys, voidaan menettää useita kaupankäyntitilaisuuksia tai jäädä mahdollisesti tappiolle, kun ohjelma ei pysty suorittamaan strategiaansa. [17]

Monitorointikomponentti hoitaa edellä mainitun sydämensykkeen tarkastelun sekä ilmoituksien lähettämisen käyttäjälle. Kaikkien järjestelmän komponenttien täytyy pystyä luomaan virhetilanteissa viesti, mikäli jotain outoa tapahtuu järjestelmän toiminnassa. Tämä viesti välitetään monitorointikomponentille, joka voi lähettää virheviestin eteenpäin esimerkiksi sähköpostilla käyttäjälle. Sähköpostiin voidaan liittää myös toimintaloki kaupankäyntiohjelman toiminnasta. [17]

Rajapinta tässä komponentissa ottaa vastaan muiden komponenttien lähettämiä ilmoituksia tai lähettää kyselyitä komponenteille niiden tilasta. Tilannetiedot talletetaan lokitiedostoihin ja monitorointikomponentti tarjoaa myös käyttöliittymälle tilannekuvaa ohjelman toiminnasta. Lisäksi tähän moduuliin voidaan asettaa muita turvatoimia. Esimerkiksi virheen sattuessa voidaan lopettaa kaupankäynti ja myydä kaikki positiot pois salkuista, jotta vältetään tappion tekeminen. Jos järjestelmässä on useita algoritmeja ja vain yksi niistä menee epäkuntoon, niin tällöin vain kyseisen algoritmin hoitamat positiot myydään pois ja lopetetaan algoritmin toiminta. Muut algoritmit voivat jatkaa toimintaansa, jos niissä ei esiinny virheitä.

4.8 Toteutus

Käyttöliittymäkomponenttia ei tässä suunnitelmassa kovin tarkasti käsitelty, mutta sen voi toteuttaa webkäyttöliittymänä, jos halutaan tarkastella järjestelmää verkossa. Toinen vaihtoehto on luoda perinteinen työpöytäkäyttöliittymä ohjelmalle. Käyttöliittymään voidaan luoda näkymiä esimerkiksi salkkujen tilanteista tai algoritmien tuotoista tai tappioista. Lisäksi käyttöliittymässä täytyy olla mahdollisuus lopettaa tiettyjen algoritmien toiminta tai koko järjestelmän toiminta. [19]

Tämän suunnitelman voi toteuttaa todennäköisesti millä tahansa olio-ohjelmointia tukevalla ohjelmointikielellä. Lähteiden perusteella algoritminen kaupankäyntiohjelma voidaan ohjelmoida kielillä Java, C++, R ja Python. Näistä kielistä Java, C++ ja Python tukevat ainakin perinteistä olio-ohjelmointia. Python on helppo ja nopea kieli ohjelmoida sekä se tukee hyvin myös koneoppimista. Lisäksi Pythoniin on olemassa valmiita kirjastoja esimerkiksi markkinadatan hakemiseen. Jos halutaan optimoida kaikki mahdollinen ohjelman toiminnassa ja maksimoida tehokkuus, kannattaa valita C++ [13]. C++ kielelle löytyviä hyödyllisiä kirjastoja ovat ainakin Boost ja QuantLib [20]. Javaa käytettäessä ohjelman voisi rakentaa Oanda Rest API:n ja Spring sovelluskehyksen (framework) päälle [17].

Aluksi valmiilla ohjelmalla kannattaa käydä niin sanottua paperikaupankäyntiä eli käydään kauppaa leikkirahalla. Siten voidaan tarkistaa, löytyykö järjestelmästä jotain ongelmia tai virheitä ilman, että menetetään pääomaa virheiden sattuessa.

Suomalaisille palveluitaan tarjoavia välittäjiä on olemassa ainakin Nordnet ja Lynx, joiden kautta kaupankäyminen onnistuu melko kattavasti ympäri maailmaa. Molemmilla näistä on olemassa API (Application Programming Interface) eli rajapinta, jonka kautta välittäjän järjestelmää voi käyttää ohjelmallisesti. Lynxillä on myös mahdollisuus paperikaupankäyntikäyttäjiin, jolloin kaupankäyntijärjestelmää voisi testata leikkirahalla ensin.

5. TULOKSET

Kaupankäyntijärjestelmälle luotiin suunnitelma komponenttitasolla, mutta joidenkin komponenttien sisäistä toteutusta suunniteltiin myös jo luokkatasolla. Suunnitelma kattaa seuraavat komponentit: datanhakija, algoritmienhallinta, toimeksiantojenhallinta, salkkujenhallinta ja järjestelmän monitorointi. Käyttöliittymä komponenttia ei suunniteltu kovin tarkasti, koska se ei ole kovin oleellinen tämän järjestelmän kannalta. Luodussa suunnitelmassa hyödynnettiin abstrakti tehdas -suunnittelumallia datanhakukomponentissa. Lähes kaikkien luokkien suunnittelussa hyödynnettiin olio-ohjelmoinnin periytymisominaisuutta. Myös SOLID -periaatteista hyödynnettiin Open-Closed -periaattetta sekä Single Responsibility -periaattetta lähes kaikissa komponenteissa. Ohjelmointikielistä Python vaikutti hyvältä vaihtoehdolta, jos halutaan nopeasti lähteä toteuttamaan suunnitelman mukaista ohjelmaa.

Aihe on todella laaja, joten kandidaatintyössä ei voitu perehtyä kaikkiin asioihin erityisen tarkasti. Suunnitelman ja kandidaatintyön pitäisi kuitenkin antaa hyvää tietoa lukijalle ja tarjota hyvän pohjasuunnitelman, josta lähteä kehittämään automaattista kaupankäyntiohjelmaa.

Tämän kandidaatintyön suunnitelmaa voidaan verrata lähteissä [6] ja [17] esiteltyihin automaattisiin kaupankäyntiohjelmiin. Nämä kaksi kaupankäyntiohjelmaa oli rakennettu kielillä R ja Java. Conlanin teoksessa *Automated Trading with R: Quantitative Research and Platform Development* luotu kaupankäyntiohjelma ei ole yhtä skaalautuva kuin mitä tähän kandityöhön on suunniteltu. Siinä ei myöskään olla käytetty olio-ohjelmointia tai suunnitteluperiaatteita vaan kirjan kaupankäyntiohjelma perustuu useisiin skripteihin, jotka lopuksi automatisoidaan. Varshneyn teoksessa *Building Trading Bots Using Java* kaupankäyntijärjestelmässä on olio-ohjelmointia ja luotiin järjestelmä, jossa voidaan hyödyntää useita eri strategioita. Kyseisestä järjestelmästä otettiin mallia joissakin tämän kandityön suunnitelman kohdissa. [6] [17]

6. YHTEENVETO

Automaattinen kaupankäyntijärjestelmä mahdollistaa pörssikaupankäynnin erilaisilla algoritmeilla ja sijoitusstrategioilla ilman ihmisen jatkuvaa ohjausta. Tällainen systeemi pystyy tekemään voittoa markkinoilla, jos siihen on lisätty toimiva kaupankäyntialgoritmi. Jotta algoritmi voidaan laittaa toimimaan, tarvitsee se ympärilleen muita tärkeitä osia kuten datalähteet, riskinhallintakomponentin ja yhteyden välittäjän palveluun.

Tässä kandidaatintyössä pohdittiin kysymystä “miten suunnitellaan algoritmiseen kaupankäyntiin soveltuva ohjelma?”. Sitä varten perehdyttiin luvussa 2 kaupankäynnin teoriaan ja luvussa 3 ohjelmistojen suunnittelun teoriaan. Luvussa 4 suunniteltiin teorian ja ohjelmistojen suunnitteluperiaatteiden pohjalta tarvittavat komponentit automaattisen kaupankäyntiohjelman toteuttamiseksi.

Olio-ohjelmointi mahdollistaa monimutkaisten ohjelmistojen toteuttamisen jakamalla suuret kokonaisuudet pienempiin osiin ja yhdistämällä nämä toisiinsa. Toteuttamalla kaupankäyntiohjelman komponentit noudattaen hyviä suunnitteluperiaatteita saavutetaan ohjelmistolle hyvä laatu ja mahdollistetaan ohjelmiston helppo muokattavuus sekä ylläpidettävyys tulevaisuudessa.

Kaupankäyntijärjestelmä suunniteltiin pitäen mielessä useat olio-ohjelmoinnin periaatteet ja suunnittelumallit. Suunnitelmasta löytyy komponenttikaavio eri komponenteista, joka osoittaa kuinka nämä komponentit liittyvät toisiinsa. Komponenttien sisäistä toimintaa ja tarkempaa suunnittelua avattiin luvun 4 alaluvuissa. Luvun 4 lopussa pohdittiin myös käytännön toteutuksen asioita kuten sopivia ohjelmointikieliä ja välittäjien rajapintoja. Tuloksena on olio-ohjelmointia hyödyntävä suunnitelma automaattiselle kaupankäyntiohjelmalle, joka pystyy skaalautumaan useammalle algoritmille sekä usean eri välittäjän palveluun.

Kandidaatintyön suunnitelma toteutettiin ottamalla mallia eri kirjoissa esitellyistä vastaavanlaisista kaupankäyntiohjelmista sekä hyödyntämällä suunnitteluperiaatteita. Kirjoissa olleet järjestelmät olivat osittain erilaisia, eivätkä ne välttämättä hyödyntäneet yhtä kattavasti suunnitteluperiaatteita ja suunnittelumalleja. Tässä työssä ei kuitenkaan aivan täysin yksityiskohtaisesti pystytty käymään kaikkia asioita läpi. Tulevaisuudessa tämän

suunnitelman pohjalta voisikin luoda toimivan ohjelman ja tutkielman, jossa käytäisiin koko ohjelman toteutus läpi yksityiskohtaisesti.

7. LÄHTEET

[1] R.T. Williams, An Introduction to Trading in the Financial Markets: Market Basics. 1 edn. US: Academic Press, 2010. Saatavissa (viitattu 22.5.2019): https://learning.oreilly.com/library/view/an-introduction-to/9780123748386/?ar

[2] R.T. Williams, An introduction to Trading in the Financial Markets: trading, markets, instruments, and processes. 1 edn. Amsterdam; Boston: Academic Press, 2011. Saatavissa (viitattu 13.5.2019): https://learning.oreilly.com/library/view/an-introduction-to/9780123748393/

[3] A. Elder, The new trading for a living: psychology, discipline, trading tools and systems, risk control, trade management. 1st edn. New York: Wiley, 2014. Saatavissa (viitattu 22.5.2019): https://learning.oreilly.com/library/view/the-new-trading/9781118963678/

[4] M. Krantz, Fundamental Analysis For Dummies, 2nd Edition. 2 edn. John Wiley & Sons, 2016. Saatavissa (viitattu 18.5.2019): https://learning.oreilly.com/library/view/fundamental-analysis-for/9781119263593/

[5] I. Ramlall, Applied Technical Analysis for Advanced Learners and Practitioners. Bingley, UNKNOWN: Emerald Group Publishing Limited, 2016. Saatavissa (viitattu 13.4.2019): http://web.a.ebscohost.com/ehost/detail/detail?vid=0&sid=a0eb22dc-8ca6-4463-9a73-8e6552aebd5d%40sdc-v-sessmgr02&bdata=JkF1dGhUeXBlP-WNvb2tpZSxpcCx1aWQmc2l0ZT1laG9zdC1saXZlJnNjb3BlPXNpdGU%3d#AN=1423583&db=nlebk

[6] C. Conlan, Automated Trading with R: Quantitative Research and Platform Development. 1 edn. Berkeley, CA: Apress L. P, 2016. Saatavissa (viitattu 18.3.2019): https://learning.oreilly.com/library/view/automated-trading-with/9781484221785/

[7] B. J. Young, K. A. Houston, J. Conallen, M. W. Engle, R. A. Maksimchuk, G. Booch, Object-Oriented Analysis and Design with Applications, Third Edition, 2007. Saatavissa (viitattu 1.6.2019): https://learning.oreilly.com/library/view/object-oriented-analysis-and/9780201895513/

[8] A. Shalloway, J. R. Trott, Design Patterns Explained: A New Perspective on Object-Oriented Design, Second Edition, 2004. Saatavissa (viitattu 1.6.2019): https://learning.oreilly.com/library/view/design-patterns-explained/0321247140/

[9] H. Zhu, Software Design Methodology: From Principles to Architectural Styles, GB: Butterworth Heinemann, 2005. Saatavissa (viitattu 18.5.2019): https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=269543

[10] W. Haoyu, Z. Haili, Basic Design Principles in Software Engineering, IEEE, 2012, s. 1251-1254. Saatavissa (viitattu 18.5.2019): https://ieeexplore-ieee-org.libproxy.tuni.fi/document/6301346

[11] J. Vlissides, R. Johnson, R. Helm, E. Gamma, Design patterns: elements of reusable object-oriented software, 1994. Saatavissa (viitattu 1.6.2019): https://learning.oreilly.com/library/view/design-patterns-elements/0201633612/

[12] C. Pauna, Automated Trading Software - Design and Integration in Business Intelligence Systems. Database Systems Journal, IX(1), 2018, s. 22–28. Saatavissa (viitattu 18.3.2019): https://doaj.org/article/35cc64b28feb47ca97473dde98fbae30

[13] E.A. Durenard, Professional automated trading: theory and practice, Hoboken, New Jersey: John Wiley & Sons, 2013. Saatavissa (viitattu 11.6.2019): https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=1411615

[14] I. Aldridge, I. Ebrary, High-frequency trading: a practical guide to algorithmic strategies and trading systems, 2nd edn. Hoboken, N.J: Wiley, 2013. Saatavissa (viitattu 18.3.2019): https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=1164806

[15] K. Kim, Electronic and algorithmic trading technology: the complete guide, Amsterdam; Boston: Academic Press, an imprint of Elsevier, 2007. Saatavissa (viitattu 18.3.2019): https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=305560

[16] K. Davey, Building algorithmic trading systems: a trader's journey from data mining to Monte Carlo simulation to live trading. Hoboken, New Jersey: Wiley, 2014. Saatavissa (viitattu 18.3.2019): https://ebookcentral.proquest.com/lib/tampere/detail.action?docID=1711618

[17] S. Varshney, Building Trading Bots Using Java, Berkeley, CA: Apress L. P., 2016. Saatavissa (viitattu 18.3.2019): https://library-books24x7-com.libproxy.tuni.fi/toc.aspx?bookid=120175

[18] C. Vinte, Upon a Message-Oriented Trading API, Informatica Economica, 14(1), 2010, s. 208–216. Saatavissa (viitattu 18.3.2019): https://search-proquest-com.libproxy.tuni.fi/docview/1433236132?accountid=14242

[19] B. Van Vliet, Building automated trading systems: with an introduction to Visual C++.NET 2005. Amsterdam; Boston: Elsevier/Academic Press, 2007. Saatavissa (viitattu 18.3.2019): https://ebookcentral.proquest.com/lib/tampere/reader.action?docID=294012

[20] C. Oliveira, Options and Derivatives Programming in C++ : Algorithms and Programming Techniques for the Financial Industry, 1 edn. Berkeley, CA: Apress L. P., 2016. Saatavissa (viitattu 18.5.2019): https://library-books24x7-com.libproxy.tuni.fi/toc.aspx?bookid=119905